

# Exercício 1:

**INSTITUTO FEDERAL MINAS GERAIS (IFMG) - CAMPUS BAMBUÍ**

**Inteligência Artificial Prof. Marcos Roberto Ribeiro**

**Lista de Exercícios 03**

Considere as estruturas de dados e algoritmos de buscas estudados.

1. Implemente uma classe **Problema** para representar problemas de buscas. A classe deve ter as seguintes características:
   1. Construtor com estado **inicial** e **meta** (opcional). Além do estado inicial e meta, a classe deve ter um atributo que informa se o custo usa heurística;
   2. Métodos abstratos **acoes(estado)** e **resultado(estado, proximo\_estado)**;
   3. Método **ativa\_heuristica()** para ativar o uso da heurística no custo;
   4. Método **heuristica(estado, proximo\_estado)** para retornar uma heurística de **estado** para

**proximo\_estado**. Nessa classe esse método pode retornar sempre 0 (zero);)

* 1. Método **custo(custo\_atual, estado, acao, proximo\_estado)** para retornar o custo de chegar ao **proximo\_estado**, com aplicação de **acao** em **estado**, partindo do **custo\_atual**
     + Inicialmente, o custo de cada passo pode ser considerando como 1 (um);
     + Se o uso de heurística estiver ativado, o custo deve somar também o valor da heurística.
  2. Método **teste\_meta(estado)** para testar se o **estado** é a meta.

1. Implemente a classe **ProblemaTeste** herdando de **Problema** para realização de comparações dos algoritmos de busca. Considere as seguintes características:
   1. O construtor deve receber o **problema** real armazená-lo como atributo da classe. O construtor deve inicializar atributos para guardar o número de estados **visitados** e **gerados** e também a solução do problema;
   2. Reimplemente os métodos **acoes()**, **custo()** e **ativa\_heuristica()** para chamar os métodos correspondentes do problema real;
   3. Sobrescreva o método **resultado(estado, acao)** para chamar o mesmo método do problema real e contabilizar os estados gerados;
   4. Reescreva o método **teste\_meta(estado)** para testar a meta com o problema real, contabilizar os estados visitados e pegar a solução (se for encontrada);
   5. Crie o seguinte método para acesso aos métodos do problema real:

def getattr (self, attr):

return getattr(self.\_problema, attr)

1

2

1. Implemente uma classe **Nodo** para representar os nós da árvore de busca considerando as seguintes características:
   1. Construtor com **estado** e, como opcional, **pai**, **acao** e **custo**. Além desses atributos, a classe deve guardar a profundidade do nó, inicializada com a profundidade do **pai** mais 1 (um);
   2. Implementar o método especial **lt (nodo)** para fazer comparações de *menor* usando o custo dos nós (necessário para ordernar os nós pelo custo);
   3. Implementar o método especial **eq (nodo)** para testar se o nó atual é igual a **nodo**

(utilize o **estado** dos nós);

* 1. Método **cria\_filho(problema, acao)** para criar um nó filho com a aplicação de **acao** no nó atual (observe os parâmetros necessários para a criação de um nó);
  2. Método **expande(problema)** para retornar a lista de filhos do nó atual com base no **pro- blema**;
  3. Métodos **caminho()** e **solucao()** para retornar o caminho do nó atual até a raiz da árvore de busca e a sequência de ações para chegar à solução (utilize o atributo com a informação do **pai**)

1. Implemente os seguintes algoritmos utilizando estruturas para armazenar a **borda** e os nós já

# explorados:

* 1. Busca em profundidade;
  2. Busca em largura;
  3. Busca de aprofundamento iterativo;
  4. Busca de custo uniforme;
  5. Busca A\* (a busca de custo uniforme ser reutilizada, depois de ativar a heurística do problema.

1. Faça uma função que receba uma lista de problemas e utilize a classe **ProblemaTeste** para testar os algoritmos de busca desenvolvidos em cada problema recebido.

# Exercício 2:

Represente o problema do aspirador de pó através de uma classe herdando da classe **Problema**. Consi- dere os seguintes pontos:

* Existe mais de um estado meta? Como tratar essa questão?
* Implemente uma heurística para usar com o algoritmo A\*. Seria correto deixar o cômodo atual sujo e ir para outro cômodo?
* Execute os algoritmos de buscas desenvolvidos partindo de um estado em que ambos cômodos estejam sujos.

# Exercício 3:

Implemente a classe **QuebraCabeca** herdando da classe **Problema** para implementar o problema do quebra cabeça deslizante de 8 peças. Considere os seguintes pontos:

* Implemente uma heurística para usar com o algoritmo A\*;
* Execute os algoritmos de buscas desenvolvidos partindo de um estado aleatório.

# Exercício 4:

Considere a resolução de problemas de busca em grafos.

1. Implemente uma classe **Grafo** para representar grafos como mapas. Considere os seguintes pontos:
   * O grafo deve permitir arestas com pesos e coordenadas para os vértices;
   * É interessante usar dicionários para representar as arestas e as coordenadas;
   * Pode ser interessante que a classe tenha métodos para retornar os vértices, coordenadas de vértices, vizinhos de vértices e peso de arestas (distância entre vértices).
2. Implemente a classe **ProblemaGrafo** herdando de **Problema** para lidar com problemas de buscas em grafos. Considere os seguintes pontos:
   * O construtor deve receber, além do estado inicial e meta, o grafo que representa o problema;
   * As ações aplicáveis a um estado são os vértices vizinhos desse estado;
   * O resultado de uma ação é a própria ação (que é um vizinho);
   * escreva o método **grafo\_heuristica(nodo)** para retornar a distância em linha reta de **nodo**

até a meta;

* + Implemente o método **custo\_grafo(custo\_atual, nodo\_a, nodo\_b)** para calcular o custo (a partir do **custo\_atual**) para chegar a **nodo\_b** a partir de **nodo\_a** (considere a possibili- dade da heurística estar ativada);
  + Sobrescreva o método **custo()** simplesmente chamando o método **custo\_grafo()**;

1. Escreva uma função para calcular a distância em linha reta entre dois vértices.
2. Implemente o problema do mapa da Romênia e execute os algoritmos de buscas desenvolvidos.